CLAIM AMENDMENTS

1. (Currently amended) In a ~~combined language compiler~~ combined language compiler product for a computer system, a method of compiling a code comprising a plurality of code statements using said ~~combined language compiler~~ combined language compiler product, said method comprising the steps of:

(a) parsing said plurality of code statements into a combined representation of said plurality of code statements;

(b) splitting said combined representation into a plurality of sets of code statements, each said set comprising a plurality of independently compilable code statements;

(c) compiling each said set of code statements; and

(d) merging each said set of compiled statements into a single executable program.

2. (Currently amended) In a ~~combined language compiler~~ combined language compiler product for a computer system, a method of compiling a code comprising a plurality of code statements using said ~~combined language compiler~~ combined language compiler product, said method comprising the steps of:

(a) parsing said plurality of code statements into a combined representation of said plurality of code statements;

(b) splitting said combined code into a plurality of sets of code statements;

(c) using at least two compilers to compile said plurality of sets of code statements; wherein each said set is compilable by one said compiler; and

(d) merging each said set of compiled statements into a single executable program.

3. (Currently amended) The method of claim 2, wherein said step (a) of parsing said plurality of code statements into said combined representation further includes the step of:

using at least two computer languages to write said ~~said~~ plurality of source code statements.

4. (Currently amended) In a combined E/C (Esterel-C) ~~language compiler~~ language compiler product for a computer system, said combined E/C ~~language compiler~~ language compiler comprising an Esterel computer language, an Esterel compiler, and a C compiler, a method of compiling of an E/C source code using said combined E/C compiler, said method comprising the steps of:

(a) parsing a plurality of statements of said E/C source code;

(b) splitting said E/C source code into a plurality of sets of code statements, each said set comprising a plurality of code statements compilable by one said compiler; and

(c) compiling each said set of code statements.

5. (Currently amended) In a combined E/C (Esterel-C) ~~language compiler~~ language compiler product for a computer system, said combined E/C ~~language compiler~~ language compiler comprising an Esterel computer language, an Esterel compiler, and a C compiler, a method of compiling of an E/C source code using said combined E/C compiler, said method comprising the ~~steps.of~~: steps of:

(a) parsing said plurality of code statements of said E/C source code into a combined representation;

(b) splitting said combined E/C code into a plurality of sets of E/C code statements;

(c) using at least two compilers to compile said plurality of sets of E/C code statements; wherein each said E/C set is compilable by one said compiler; and

(d) merging each said set of compiled E/C statements into a single executable program.

6. (Original) The method of claim 5, wherein said step (a) of parsing said plurality of E/C code statements into said combined E/C code further includes the step of:

using at least two computer languages, said Esterel computer language and said C computer language, to write said plurality of E/C source code statements.

7. (Original) The method of claim 5, wherein said step (a) of parsing said plurality of E/C code statements further includes the step of:

adding a set of modified Esterel reactive statements into said C language.

8. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel await statement:

await sig;

with the ECL statement:

await(sig).

9. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel pausing statement selected from the group consisting of:

{await tick, and pause}

with the ECL statement:

await().

10. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel emit statement:

emit sig;

with the ECL statement:

emit(sig).

11. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel emit with value statement:

emit sig(val);

with the ECL statement:

emit(sig, val).

12. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel get value of signal syntax:

?sig

with the ECL syntax:

sig.

13. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel signal testing statement:

present signal_expression then

stmt1

else

stmt2

with the ECL statement:

present (signal_expression) stmt1;

else stmt2;.

14. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel signal operator:

{and}

with the ECL syntax:

{&).

15. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel signal operator:

{or}

with the ECL syntax:

{ | }.

16. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel signal operator:

{not}

with the ECL syntax:

{ ~ }.

17. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel abortion preemption statement:

```
abort
        stmt1;
when signal _ expression do
        stmt2;
end abort
```

with the ECL abortion preemption statement:

```
do stmt1;
```

abort(signal_expression);

handle stmt2.

18. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel weak abortion preemption statement:

weak abort

stmt1;

when signal_expression do

stmt2;

end

with the ECL weak abortion preemption statement:

do stmt1;

weak_abort (signal_expression);

handle stmt2.

19. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel suspension preemption statement:

suspend

stmt;

when signal_expression

with the ECL 'suspension preemption' statement:

```
do stmt;
suspend (signal_expression);.
```

20. (Original) The method of claim 7, wherein said step of adding said set of modified Esterel reactive statements into said C language further includes the step of:

replacing

the Esterel concurrency statement:

```
stmt1 || stmt2
```

with the ECL concurrency statement selected from the group consisting of:

```
{fork
    stmt1;
        stmt2;
    join;
```

and

```
par stmt1;
par stmt2;}.
```

21. (Original) The method of claim 4 further comprising the step of:

(d) checking whether each said set of code statements satisfies the semantics of each said language.

22. (Original) The method of claim 4 further comprising the step of:

(e) merging each said compiled code statement into a single E/C executable program.

23. (Original) The method of claim 4, wherein said step (b) of splitting said E/C code into said plurality of sets of code statements further comprises the steps of:

splitting said E/C code at a specification level into two different trial E/C codes, wherein said first trial E/C code includes a first plurality of E/C code modules including a first plurality of internal module computations, a first plurality of inter-module communications, and a first level of reactivity, and wherein said second trial E/C code includes a second plurality of E/C code modules including a second plurality of internal module computations, a second plurality of inter-module communications, and a second level of reactivity;

comparing said first trial E/C code with said second trial E/C code;

assessing the difference in compilation time and the difference in execution time between said two trial E/C codes; and

selecting an optimum trial E/C code.

24. (Original) The method of claim 4, wherein said step (b) of splitting said E/C code into said plurality of sets of code statements further comprises the step of:

(b1) splitting said E/C code at a compilation level into a plurality of reactive code statements and into a plurality of non-reactive code statements, wherein each said reactive code statement is assigned to be compiled by an Esterel compiler;

and wherein each said non-reactive code statement is assigned to be compiled by said C compiler.

25. (Original) The method of claim 4, wherein said step (b) of splitting said E/C code into said plurality of sets of code statements further comprises the step of:

(b1) using a meta-optimization loop for splitting said E/C code at a compilation level into a plurality of reactive code statements and into a plurality of non-reactive code statements, wherein each said reactive code statement is assigned to be compiled by an Esterel compiler; and

wherein each said non-reactive code statement is assigned to be compiled by said C compiler.

26. (Original) The method of claim 25, wherein said step (b1) of using said meta-optimization loop for splitting said E/C code further includes the steps of:

(b1,1) splitting said E/C code into a plurality of reactive code statements and into a plurality of non-reactive code statements;

(b1,2) checking whether said splitting step (b1,1) optimizes said E/C compilation process; and

(b1,3) if said E/C compilation process is not optimized, repeating said steps (b1,1) and (b1,2) until said E/C compilation process is optimized.

27. (Original) The method of claim 24, wherein said step (b1) of splitting said E/C code at said compilation level into said plurality of reactive code statements and into said plurality of non-reactive code statements further includes the steps of:

(b1,1) parsing said E/C source code to create a parse tree including said E/C source code;

(b1,2) analyzing said E/C source code parse tree by traversing said parse tree and by determining for each said E/C code statement whether it is a part of an Esterel code file or a part of a C code file; and

(b1, 3) writing said E/C source code including said Esterel file and said C file.

28. (Original) The method of claim 27, wherein said step (b1,2) of analyzing said E/C source code further includes the steps of:

placing each said E/C locally reactive code statement in said Esterel file;

placing each said E/C locally non-reactive code statement in said C file; and

globally analyzing each said non-local E/C code statement.

29. (Original) The method of claim 27, wherein said step (b1,2) of analyzing said E/C source code further includes the steps of:

placing an E/C code loop including a halting statement in said Esterel file; and

placing an E/C code loop lacking a halting statement in said C file.

30. (Original) The method of claim 27, wherein said step (b1,2) of analyzing said E/C source code further includes the step of:

placing in said Esterel file a reactive statement selected from the group consisting of: {signal waiting, signal emission, signal testing, preemption, and concurrency}.

31. (Original) The method of claim 27, wherein said step (b1,2) of analyzing said E/C source code further includes the step of:

placing in said C file a construct defining and manipulating data type, wherein said construct is selected from the group consisting of: {type definition, field access, and pointer access}.

32. (Original) The method of claim 4, wherein said step (b) of splitting said E/C code into said plurality of sets of code statements further comprises the step of:

splitting said E/C code based on an implementation method.

33. (Original) The method of claim 32, wherein said step of splitting said E/C code based on said implementation method further includes the steps of:

reclassifying a reactive statement as being compilable by either said Esterel compiler or by said C compiler if said implementation method is substantially sufficient to process said reactive statement.

34. (Original) The method of claim 21, wherein said step (d) of checking whether each said compiled code statement satisfies the semantics of each said language further includes the steps of:

using said Esterel compiler to find and analyze a causality problem; and

reporting said causality problem to the user at the E/C level.

35. (Original) The method of claim 34, wherein said step of using said Esterel compiler to find and analyze said causality problem further includes the step of:

checking by said Esterel compiler whether a set of reactive code statements includes a finite state machine equivalent that can be analyzed statically.

36. (Original) The method of claim 22, wherein said step (e) of merging each said compiled code statement into said single E/C executable program further includes the step of:

merging a C-compiled code into an Esterel-compiled (E-compiled) code by calling a procedure from said E-compiled code.

37. (Original) The method of claim 36 further including the step of:

implementing said procedure call from said E-compiled code as a macro in C.

38. (Original) The method of claim 36 further including the step of: implementing said procedure call from said E-compiled code by passing variables by reference to said procedure.

39. (Currently amended) In a ~~combined language compiler~~ combined language compiler product for a computer system, a method of compiling of a hybrid source code using said combined language- compiler, said method comprising the steps of:

(a) parsing a plurality of statements of said hybrid source code;

(b) splitting said hybrid source code into a plurality of sets of code statements, each said set comprising a plurality of code statements compilable by one said compiler;

(c) compiling each said set of code statements; and

(d) merging each said compiled code statement into a single executable program.

40. (Currently amended) In a ~~combined language compiler~~ combined language compiler product for a computer system, a method of compiling of a hybrid source code using said combined compiler, said method comprising the steps of:

(a) parsing said plurality of code statements of said hybrid source code into a combined representation;

(b) splitting said combined representation into a plurality of sets of hybrid code statements;

(c) using at least two compilers to compile said plurality of sets of hybrid code statements; wherein each said set is compilable by one said compiler; and

(d) merging each said set of compiled statements into a single executable program.

41. (Original) The method of claim 40, wherein said step (a) of parsing said plurality of hybrid code statements further includes the step of:

using at least two computer languages to write said plurality of hybrid source code statements.

42. (Original) The method of claim 40, wherein said step (b) of splitting said hybrid code into said plurality of sets of code statements further comprises the steps of:

splitting said hybrid code at a specification level into two different trial codes, wherein said first trial code includes a first plurality of code modules including a first plurality of internal module computations, a first plurality of inter-module communications, and a first level of reactivity, and wherein said second trial code includes a second plurality of code modules including a second plurality of internal module computations, a second plurality of inter-module communications, and a second level of reactivity;

comparing said first trial code with said second trial code;

assessing the difference in compilation time and the difference in execution time between said two trial codes; and

selecting an optimum trial code.

43. (Currently amended) The method of claim 40, wherein said step (b) of splitting said hybrid code further includes the step of:

~~slitting~~ splitting said hybrid code at a compilation level into a plurality of reactive code statements and a plurality of non-reactive code statements.

44. (Original) The method of claim 40, wherein said step (b) of splitting said hybrid code further comprises the step of:

splitting said hybrid code based on an implementation method.

45. (Currently amended) A ~~combined language compiler~~ combined language compiler product for a computer system, comprising;

(a) a merged syntax defining a plurality of acceptable code statements;

(b) a splitter configured to split said combined code into a plurality of sets of code statements;

(c) a compiler configured to compile each said set of code statements; and

(d) a post-compiler level merger configured to merge each said compiled code statement into a single executable program.

46. (Currently amended) A combined hybrid ~~language compiler~~ language product for a computer system, comprising:

(a) a syntax of a hybrid language defining a plurality of acceptable code statements;;

(b) a splitter configured to split said hybrid code into a plurality of sets of code statements, each said set comprising a plurality of code statements, each said code statement compilable independently;

(c) a compiler configured to compile each said set of code statements;

(d) a checker configured to check whether each said compiled code statement satisfies the semantics of said hybrid language; and

(e) a post-compiler level merger configured to merge each said compiled code statement into a single executable program.

47. (Currently amended) A computer-usable apparatus useful in association with a ~~combined language compiler~~ combined language compiler, said ~~combined language compiler~~ combined language compiler configured to compile a plurality of code statements; said computer-usable apparatus including computer-readable code instructions configured to cause said ~~combined language compiler~~ combined language compiler to execute the steps of:

(a) defining a plurality of acceptable statements of a combined code;

(b) splitting said combined code into a plurality of sets of code statements, each said set comprising a plurality of code statements compilable independently;

(c) compiling each said set of code statements; and

(d) merging each said compiled code statement into a single executable program.

48. (Original) A computer data signal embodied in a carrier wave comprising:

(a) a first merged source code segment comprising a plurality of code statements; wherein each said code statement is configured to be compiled independently;

(b) a split source code segment comprising a plurality of sets of code statements;

(c) a combined compiled source code segment comprising a plurality of compiled source code segments; and

(d) a second merged source code segment comprising a plurality of compiled code statements as a single executable program.

49. (Original) An E/C computer data signal embodied in an E/C carrier wave comprising:

(a) an E/C pre-compiler-merged source code segment comprising a plurality of code statements, each said code statement written using a C computer language, or an Esterel computer language;

(b) a split E/C source code segment comprising a plurality of sets of code statements, each said set comprising a plurality of code statements compilable independently;

(c) an E/C compiled source code segment comprising a plurality of compiled source code segments; and

(d) an E/C post-compiler-merged source code segment comprising a plurality of compiled code statements as a single E/C executable program.

50. (Currently amended) A computer-readable code embedded in a storage medium, wherein said computer readable code is manipulated by a ~~combined language compiler~~ combined language compiler, ~~combined language compiler~~ combined language compiler configured to compile a plurality of code statements written using a plurality of computer languages, said ~~combined language compiler~~ combined language compiler configured to execute the steps of:

(a) accepting a combined code comprising a plurality of code statements;

(b) splitting said combined code into a plurality of sets of code statements, each said set comprising a plurality of independently compilable code statements;

(c) compiling each said set of code statements; and

(d) merging each said compiled code statement into a single executable program.

51. (Currently amended) A computer-readable code embedded in a storage medium, wherein said computer readable code is manipulated by a combined Esterel-C (E/C) language ~~language compiler~~ language compiler, said combined E/C ~~language compiler~~ language compiler comprising an E/C language, an Esterel compiler, and a C compiler, said combined E/C ~~language compiler~~ language compiler configured to compile a plurality of code statements written using said E/C language, said combined E/C ~~language compiler~~ language compiler configured to execute the steps of:

(a) accepting a plurality of statements of an E/C source code;

(b) splitting said E/C source code into a plurality of sets of code statements, each said set comprising a plurality of independently compilable code statements;

(c) compiling each said set of code statements;

(d) checking whether each said compiled code statement satisfies the semantics of each said language; and

(e) merging each said compiled code statement into a single E/C executable program.